BSC-IT

SEM -5 Enterprise Java

NOV-2018 Q.P.Code:57841

Q1 Attempt any three of the following: (15)

a) Explain the architecture of Java Enterprise Application. (5)

• **java EE Application Components**:

Java EE applications are made up of components. A Java EE component is a selfcontained functional software unit that is assembled into a Java EE application with its related classes and files and communicates with other components. The Java EE specification defines the following Java EE components:

- Application clients and applets are client components.
- Java Servlet and Java Server Pages (JSP) technology components are web components.
- Enterprise JavaBeans (EJB) components (enterprise beans) are business components. Java EE components are written in the Java programming language and compiled in the same way as any Java programming language program. The difference when you work with the Java EE platform, is Java EE components are assembled into a Java EE application, verified that they are well-formed and in compliance with the Java EE specification, and deployed to production where they are run and managed by the Java EE server.

Java EE Architecture:

Normally, thin-client multitiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details. The component-based and platform-independent Java EE architecture makes Java EE applications easy to write because business logic is organized into reusable components and the Java EE server provides underlying services in the form of a container for every component type. Because you do not have to develop these services yourself, you are free to concentrate on solving the business problem at hand.

Containers and Services:

Component are installed in their containers during deployment and are the interface between a component and the low-level platform-specific functionality that supports the component. Before a web, enterprise bean, or application client component can be executed, it must be assembled into a Java EE application and deployed into its container.

The assembly process involves specifying container settings for each component in the Java EE application and for the Java EE application itself. Container settings customize the underlying support provided by the Java EE Server, which include services such as security, transaction management, Java Naming and Directory Interface (JNDI) lookups, and remote connectivity. Here are some of the highlights:

- The Java EE security model lets you configure a web component or enterprise bean so system resources are accessed only by authorized users.

- The Java EE transaction model lets you specify relationships among methods that make up a single transaction so all methods in one transaction are treated as a single unit.

Container Types:

The deployment process installs Java EE application components in the following types of Java EE containers.

- An Enterprise JavaBeans (EJB) container manages the execution of all enterprise beans for one Java EE application. Enterprise beans and their container run on the Java EE server.
- A web container manages the execution of all JSP page and servlet components for one Java EE application. Web components and their container run on the Java EE server.
- An application client container manages the execution of all application client components for one Java EE application. Application clients and their container run on the client machine.
- An applet container is the web browser and Java Plug-in combination running on the client machine.

b) Classify the EE Containers. Explain. **(5)**

Java EE Containers:

Normally, thin-client multitier applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details. The component-based and platform-independent Java EE architecture makes Java EE applications easy to write because business logic is organized into reusable components. In addition, the Java EE server provides underlying services in the form of a container for every component type. Because you do not have to develop these services yourself, you are free to concentrate on solving the business problem at hand.

Container Services:

Containers are the interface between a component and the low-level platform specific functionality that supports the component. Before a web, enterprise bean, or application client component can be executed, it must be assembled into a Java EE module and deployed into its container. The Java EE security model lets you configure a web component or enterprise bean so that system resources are accessed only by authorized users.

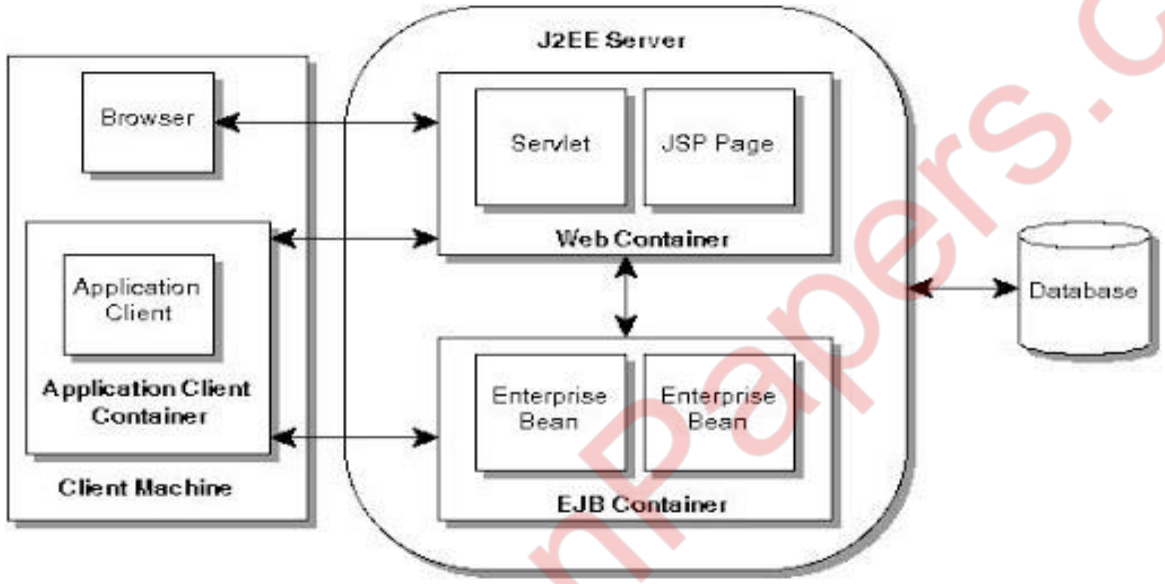
- The Java EE transaction model lets you specify relationships among methods that make up a single transaction so that all methods in one transaction are treated as a single unit.
- JNDI lookup services provide a unified interface to multiple naming and directory services in the enterprise so that application components can access these services.
- The Java EE remote connectivity model manages low-level communications between clients and enterprise beans. After an enterprise bean is created, a client invokes methods on it as if it were in the same virtual machine.

Container Types:

- **Java EE server:** The runtime portion of a Java EE product. A Java EE server provides EJB and web containers.

- **Enterprise JavaBeans (EJB) container:** Manages the execution of enterprise beans for Java EE applications. Enterprise beans and their container run on the Java EE server.
- **Web container:** Manages the execution of JSP page and servlet components for Java EE applications. Web components and their container run on the Java EE server.
- **Application client container:** Manages the execution of application client components. Application clients and their container run on the client.
- **Applet container:** Manages the execution of applets. Consists of a web browser and Java Plug-in running on the client together.

The deployment process installs Java EE application components in the Java EE containers as illustrated in following Figure:



c) Write a short note on javax.servlet package **(5)**

Servlet API: The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api. The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The javax.servlet.http package contains interfaces and classes that are responsible for http requests only. Let's see what are the interfaces of javax.servlet package. Interfaces in javax.servlet package There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletConfig
3. ServletRequest
4. ServletContext
5. ServletResponse
6. SingleThreadModel
7. RequestDispatcher
8. Filter

• **Classes in javax.servlet package**

There are many classes in javax.servlet package. They are as follows:

1.GenericServlet
2.ServletContextEvent

3.ServletInputStream
4.ServletRequestAttributeEvent
5.ServletOutputStream
6.ServletContextAttributeEvent
7.ServletRequestWrapper
8.ServletException
9.ServletResponseWrapper
10. ServletRequestEvent ServletRequestEvent

• **Interfaces in javax.servlet.http package:**

There are many interfaces in javax.servlet.http package. They are as follows:

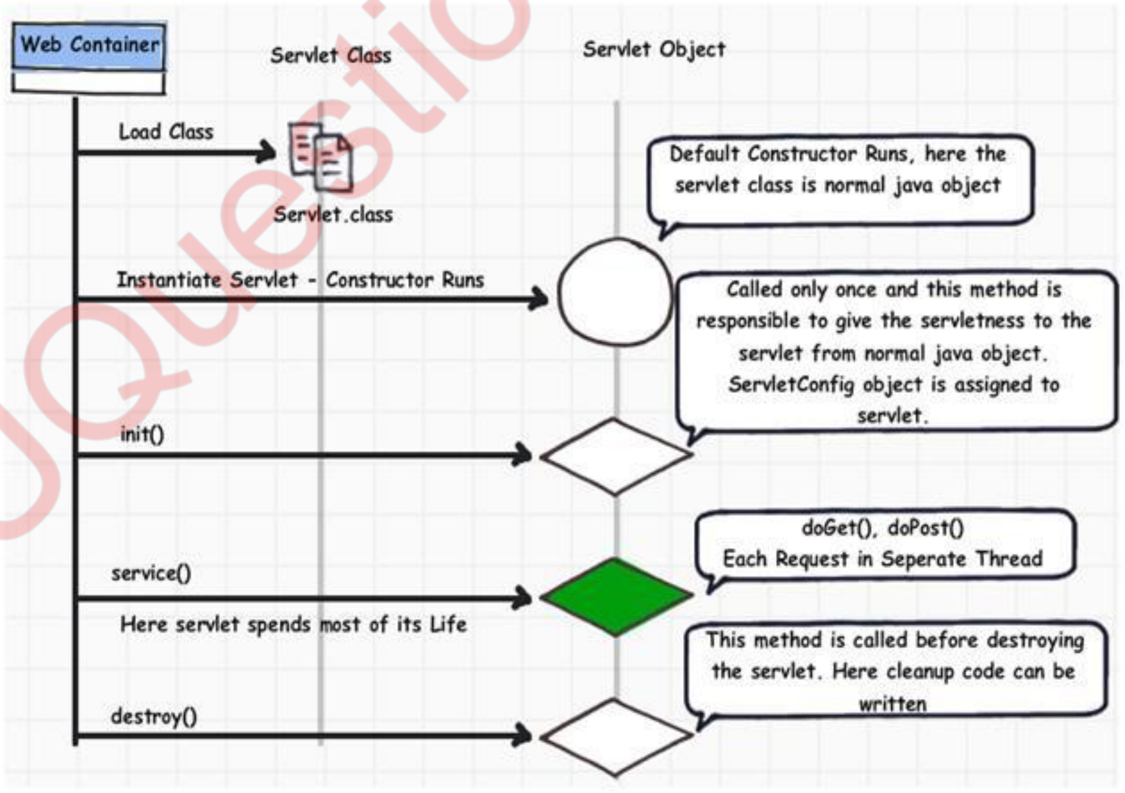
1. HttpServletRequest
2. HttpSessionAttributeListener
3. HttpServletResponse
4. HttpSessionBindingListener
5. HttpSession
6. HttpSessionActivationListener
7. HttpSessionListener

• **Classes in javax.servlet.http package**:

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent

d)Explain the lifecycle of a servlet application. **(5)**


Web Container
Servlet Class
Servlet Object
Load Class
Servlet.class
Default Constructor Runs, here the servlet class is normal java object
Instantiate Servlet - Constructor Runs
Called only once and this method is responsible to give the servletness to the servlet from normal java object. ServletConfig object is assigned to servlet.
init()
doGet(), doPost()
Each Request in Seperate Thread
service()
Here servlet spends most of its Life
This method is called before destroying the servlet. Here cleanup code can be written
destroy()

It describes how a servlet is :

- Loaded
- Instantiated
- Initialized
- Services requests
- Destroyed
- Finally garbage collected

Initialization:

- In order to initialize a servlet, the servlet engine first locates its class.
- The servlet engine uses the usual java class loading facilities to load the servlet class into the JVM.
- Once loaded, the servlet engine instantiates an instance of that servlet.
- Then it calls the init(ServletConfig config) method.
- init() is called immediately after the server constructs the object.
- Declaration : public void init(SerlvetConfig config) throws ServletException
- During initialization , servlet has access to two objects:
- ServletConfig and ServletConext
- Following tasks can be implemented by overridding init() of Servlet:
- Reading initializing parameters using ServletConfig object.
- Reading configuration data from persistent resources
- Initializing a database driver
- Opening a JDBC connection
- Writing log information

Runtime:

- After the server loads and initializes the servlet, it is able to handle client requests.
- It process them in service() method.
- Each client request is run in separate thread.
- Declaration:
- public void service(ServletRequest req, ServletResponse res) throws ServletException,IOException
- Request object helps in accessing the original request data and response object provides methods that helps the servlet to build a response.

Destruction:

- When instructed to unload the servlet, servlet engine calls the detroy().
- Declaration : public void destroy()
- The method runs once.
- Tasks performed by the method are:
- Synchronizing cleanup tasks
- Informing other application that servlet is no longer in service.

e) Explain the architecture of JDBC. **(5)**

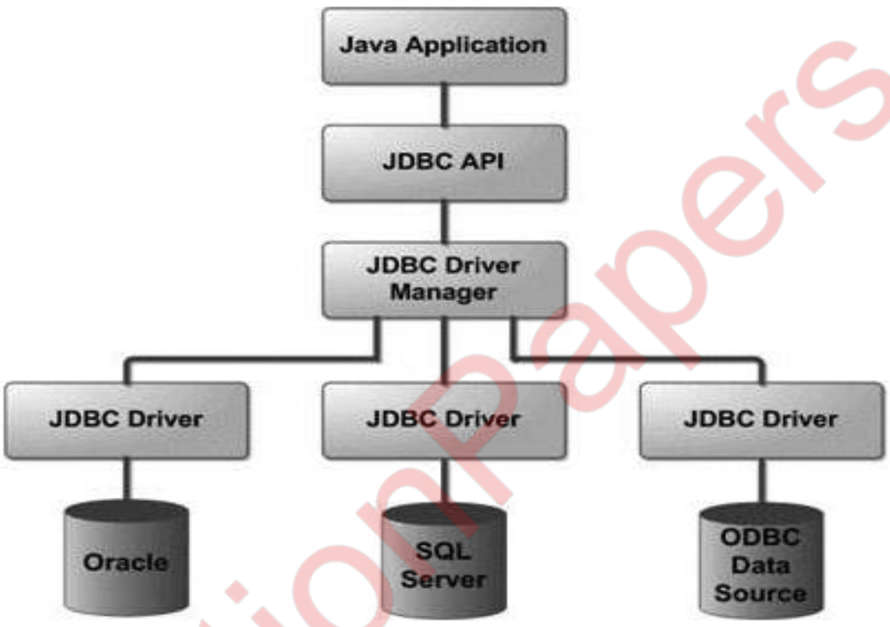
JDBC Architecture :

The JDBC API supports both two-tier and three-tier processing models for database access but in general, JDBC Architecture consists of two layers

• **JDBC API:** This provides the application-to-JDBC Manager connection.

• **JDBC Driver API:** This supports the JDBC Manager-to-Driver Connection.

The JDBC API uses a driver manager and database-specific drivers to provide transparent connectivity to heterogeneous databases. The JDBC driver manager ensures that the correct driver is used to access each data source. The driver manager is capable of supporting multiple concurrent drivers connected to multiple heterogeneous databases. Following is the architectural diagram, which shows the location of the driver manager with respect to the JDBC drivers and the Java application



f) Explain the architecture of Java Enterprise Application. **(5)**

• **java EE Application Components**:

Java EE applications are made up of components. A Java EE component is a selfcontained functional software unit that is assembled into a Java EE application with its related classes and files and communicates with other components. The Java EE specification defines the following Java EE components:

- Application clients and applets are client components.
- Java Servlet and JavaServer Pages (JSP) technology components are web components.
- Enterprise JavaBeans (EJB) components (enterprise beans) are business components. Java EE components are written in the Java programming language and compiled in the same way as any Java programming language program. The difference when you work with the Java EE platform, is Java EE components are assembled into a Java EE application, verified that they are well-formed and in compliance with the Java EE specification, and deployed to production where they are run and managed by the Java EE server.

Java EE Architecture:

Normally, thin-client multitiered applications are hard to write because they involve many lines of intricate code to handle transaction and state management, multithreading, resource pooling, and other complex low-level details. The component-based and platform-independent Java EE architecture makes Java EE applications easy to write because business logic is organized into reusable components and the Java EE server provides underlying services in the form of a container for every component type. Because you do not have to develop these services yourself, you are free to concentrate on solving the business problem at hand.

Containers and Services:

Component are installed in their containers during deployment and are the interface between a component and the low-level platform-specific functionality that supports the component. Before a web, enterprise bean, or application client component can be executed, it must be assembled into a Java EE application and deployed into its container.

The assembly process involves specifying container settings for each component in the Java EE application and for the Java EE application itself. Container settings customize the underlying support provided by the Java EE Server, which include services such as security, transaction management, Java Naming and Directory Interface (JNDI) lookups, and remote connectivity. Here are some of the highlights:

- The Java EE security model lets you configure a web component or enterprise bean so system resources are accessed only by authorized users.
- The Java EE transaction model lets you specify relationships among methods that make up a single transaction so all methods in one transaction are treated as a single unit.

Container Types:

The deployment process installs Java EE application components in the following types of Java EE containers.

- An Enterprise JavaBeans (EJB) container manages the execution of all enterprise beans for one Java EE application. Enterprise beans and their container run on the Java EE server.
- A web container manages the execution of all JSP page and servlet components for one Java EE application. Web components and their container run on the Java EE server.
- An application client container manages the execution of all application client components for one Java EE application. Application clients and their container run on the client machine.
- An applet container is the web browser and Java Plug-in combination running on the client machine.

Q2 Attempt any three of the following: (15)

a) Explain the importance of RequestDispatcher Interface in javax.servlet. Add suitable example to explain the methods of the interface. (5)

- It is used for dispatching a request.
- It encapsulates a reference to a web resource located at the path specified within the scope of the same servlet context.

- It defines an object, which receives requests from a visitor's browser and sends the requests to any other resource on the web server.
- The servlet engine creates the RequestDispatcher object, which is used as a wrapper around a web server resource located through the path specified or given by a particular name.
- We can get the object of RequestDispatcher interface by using the method: getRequestDispatcher(), there are two ways of using this method:
  javax.servlet.ServletContext.getRequestDispatcher()
  javax.servlet.ServletRequest.getRequestDispatcher()

Methods of RequestDispatcher :
This interface has two methods:
include() : it allows including the content produced by another resource such as servlet, JSP or HTML file in the calling servlet's response.
forward() : it allows forwarding the request to another servlet, a JSP or HTML page on the server. This resource then take over the responsibility for producing the response.
It is useful when one servlet does some preliminary processing of a request and wants to let another object complete the generation of the response.

index.html
```
<form action="servlet1" method="post">
 Name:<input type="text" name="userName"/><br/>
 Password:<input type="password" name="userPass"/><br/>
<input type="submit" value="login"/>
 </form>
```
Login.java
```
 import java.io.*;
 import javax.servlet.*;
 import javax.servlet.http.*;

public class Login extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {

 response.setContentType("text/html");
 PrintWriter out = response.getWriter();

String n=request.getParameter("userName");
String p=request.getParameter("userPass");

 if(p.equals("servlet"){
 RequestDispatcher rd=request.getRequestDispatcher("servlet2");
rd.forward(request, response);
 }
else{
out.print("Sorry UserName or Password Error!");
RequestDispatcher rd=request.getRequestDispatcher("/index.html");
```

```
 rd.include(request, response);

 }
 }
 }
```

WelcomeServlet.java

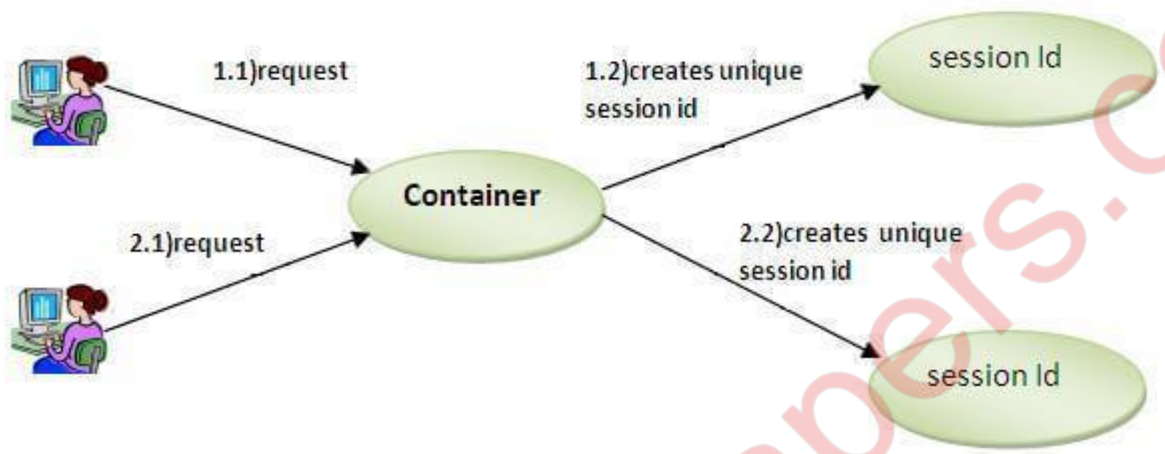
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class WelcomeServlet extends HttpServlet {
public void doPost(HttpServletRequest request, HttpServletResponse response
)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String n=request.getParameter("userName");
out.print("Welcome "+n);
}
}
```

web.xml

```
<web-app>
<servlet>
 <servlet-name>Login</servlet-name>
<servlet-class>Login</servlet-class>
</servlet>
<servlet>
<servlet-name>WelcomeServlet</servlet-name>
<servlet-class>WelcomeServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>Login</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>WelcomeServlet</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
<welcome-file-list>
<welcome-file>index.html</welcome-file>
</welcome-file-list>
  </web-app>
```

b) Explain how java handles session management of a servlet application through HTTP Session. (5)

HttpSession: interface In such case, container creates a session id for each user.The container uses this id to identify the particular user.An object of HttpSession can be used to perform two tasks:
1. bind objects
2. view and manipulate information about a session, such as the session identifier, creation time, and last accessed time.



index.html
```
 <form action="servlet1">
 Name:<input type="text" name="userName"/><br/>
 <input type="submit" value="go"/>
 </form>
FirstServlet.java
import java.io.*;
 import javax.servlet.*;
 import javax.servlet.http.*;


 public class FirstServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response){
try{

 response.setContentType("text/html");
 PrintWriter out = response.getWriter();

String n=request.getParameter("userName");
 out.print("Welcome "+n);

 HttpSession session=request.getSession();
session.setAttribute("uname",n);
```

```
 out.print("<a href='servlet2'>visit</a>");

 out.close();

}catch(Exception e){System.out.println(e);}
 }

 }
```

SecondServlet.java

```
 import java.io.*;
 import javax.servlet.*;
 import javax.servlet.http.*;

 public class SecondServlet extends HttpServlet {

public void doGet(HttpServletRequest request, HttpServletResponse response)
try{

 response.setContentType("text/html");
PrintWriter out = response.getWriter();
HttpSession session=request.getSession(false);
 String n=(String)session.getAttribute("uname");
out.print("Hello "+n);
out.close();
}catch(Exception e){System.out.println(e);}
}
}
<web-app>
<servlet>
 <servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s2</servlet-name>
```

```
        <url-pattern>/servlet2</url-pattern>
        </servlet-mapping>
</web-app>
```

c) What is cookie? Explain the need of the creation of the same. Explain the methods of cookie class that facilitative the creation, manipulation and deletion of the cookies. (5)

COOKIE:

- HTTP does not maintain a persistent connection. Each request made by a web browser is made using a new connection. Hence this protocol does not maintain state across HTTP requests.
- This causes a problem for websites attempting to do business on the Internet.
- To overcome this issue, somehow the web server must be able to register that the multiple requests which are coming are originated from the very same browser.
- Here Cookies came in.
- Cookies are small text file that arte created by a server side program such as a servlet run at the web server.

Cookies in Servlet:
A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.
Useful Methods of Cookie class:
There are given some commonly used methods of the Cookie class.

- **public void setMaxAge(int expiry):** Sets the maximum age of the cookie in seconds.
- **public String getName():** Returns the name of the cookie. The name cannot be changed after creation.
- **public String getValue():** Returns the value of the cookie.
- **public void setName(String name):** changes the name of the cookie.
- **public void setValue(String value):** changes the value of the cookie.
- **public void addCookie(Cookie ck):**method of HttpServletResponse interface is used to add cookie in response object.
- **public Cookie[] getCookies():**method of HttpServletRequest interface is used to return all the cookies from the browser.

Delete Cookies with Servlet:

To delete cookies is very simple. If you want to delete a cookie then you simply need to follow up following three steps –

- Read an already existing cookie and store it in Cookie object.
- Set cookie age as zero using setMaxAge() method to delete an existing cookie
- Add this cookie back into response header

d) What is Non-Blocking I/O? Explain WriteListener and ReadListener performing in Non-Blocking I/O? (5)

- **Nonblocking I/O Support in javax.servlet.ServletInputStream**

Methods	Description
void setReadListener(ReadListener rl)	Associates this input stream with a listener object that contains callback methods to read data asynchronously. You provide the listener object as an anonymous class or use another mechanism to pass the input stream to the read listener object.
boolean isReady()	Returns true if data can be read without blocking.
boolean isFinished()	Returns true when all the data has been read.

- **Nonblocking I/O Support in javax.servlet.ServletOutputStream**

Methods	Description
void setWriteListener(WriteListener wl)	Associates this output stream with a listener object that contains callback methods to write data asynchronously. You provide the write listener object as an anonymous class or use another mechanism to pass the output stream to the write listener object.
boolean isReady()	Returns true if data can be written without blocking

- **Listener Interfaces for Nonblocking I/O Support**

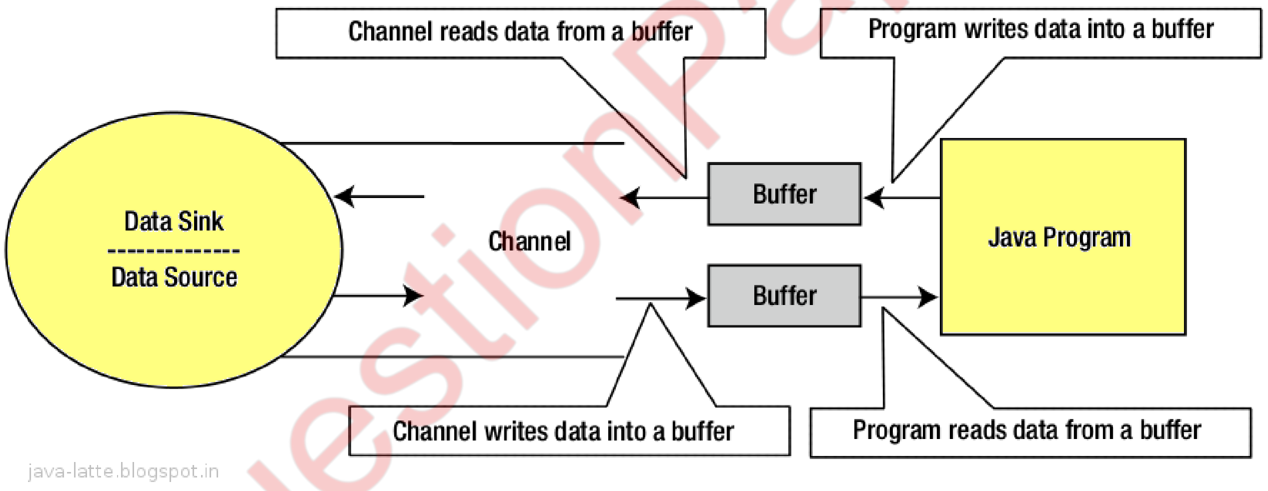
Interface	Methods	Description
ReadListener	void onDataAvailable() void onAllDataRead() void onError(Throwable t)	A ServletInputStream instance calls these methods on its listener when there is data available to read, when all the data has been read, or when there is an error
WriteListener	void onWritePossible() void onError(Throwable t)	A ServletOutputStream instance calls these methods on its listener when it is possible to write data without blocking or when there is an error.

e) Explain the working of Non-Blocking I/O. (5)

Non blocking I/O:

- It is Buffer-oriented
- Channels are available for Non-blocking I/O operation
- Selectors are available for Non-blocking I/O operation
- Non blocking IO does not wait for the data to be read or write before returning.
- Java NIO non- blocking mode allows the thread to request writing data to a channel, but not wait for it to be fully written. The thread is allowed to go on and do something else in a mean time.
- java NIO is buffer oriented I/O approach. Data is read into a buffer from which it is further processed using a channel. In NIO we deal with the channel and buffer for I/O operation.
- The major difference between a channel and a stream is:
  - A stream can be used for one-way data transfer.
  - A channel provides a two-way data transfer facility.

Therefore with the introduction of channel in java NIO, the non-blocking I/O operation can be performed. Let's see the interaction between channel, buffers, java program, data source and data sink:


Channel reads data from a buffer
Program writes data into a buffer
Data Sink
Data Source
Channel
Buffer
Buffer
Java Program
Channel writes data into a buffer
Program reads data from a buffer
java-latte.blogspot.in

Interaction between a channel, buffers, a Java program, a data source, and a data sink

Channels:

In Java NIO, the channel is a medium that transports the data efficiently between the entity and byte buffers. It reads the data from an entity and places it inside buffer blocks for consumption. Channels act as gateway provided by java NIO to access the I/O mechanism. Usually channels have one-to-one relationship with operating system file descriptor for providing the platform independence operational feature.

f) Explain about the file uploading feature of a servlet.
(5)

A Servlet can be used with an HTML form tag to allow users to upload files to the server. An uploaded file could be a text file or image file or any document.

Creating a File Upload Form:

The following HTM code below creates an uploader form. Following are the important points to be noted down –

- The form method attribute should be set to POSTmethod and GET method can not be used
- The form enctype attribute should be set to multipart/form-data
- The form action attribute should be set to a servlet file which would handle file uploading at backend server. Following example is using UploadServlet servlet to upload file.
- To upload a single file you should use a single <input .../> tag with attributetype="file". To allow multiple files uploading, include more than oneinput tags with different values for the name attribute. The browserassociates a Browse button with each of them.

Q3 Attempt any three of the following: (15)

a)What are directives in JSP? Explain page directive in detail. (With all its attributes) (5)

Directives

- JSP directives serve special processing information about the page to the JSP server.
- These are used to:
  - Set global values such as class declaration
  - Methods to be implemented
  - The scripting language for the page and so on
- They do not produce any output that is visible to the client.
- They are typically used to provide direction to the JSP server processing the page.
- Syntax:

<%@ directive name attributes="value"%>

- Following are the different types of directives
  - The page directive
  - The include directive
  - The taglib directive

The page directive:

- It can be placed anywhere within the document, but often placed at the very top.
- It supports several attributes that affect the whole page.
- A single page can contain multiple page directives.
- The page directive is used to provide instructions to the container. These instructions pertain to the current JSP page.
- You may code page directives anywhere in your JSP page. By convention, page directives are coded at the topof the JSP page.

Following is the basic syntax of the page directive –
<%@ page attribute = "value" %>
Attributes Following table lists out the attributes associated with the page directive –

1 buffer: Specifies a buffering model for the output stream.
2 autoFlush: Controls the behavior of the servlet output buffer.
3 contentType:Defines the character encoding scheme.
4 errorPage: Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
5 isErrorPage: Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
6 extends: Specifies a superclass that the generated servlet must extend.
7 import: Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
8 info: Defines a string that can be accessed with the servlet's getServletInfo() method.
9 isThreadSafe: Defines the threading model for the generated servlet.
10 language: Defines the programming language used in the JSP page.
11 session: Specifies whether or not the JSP page participates in HTTP sessions
12 isELIgnored: Specifies whether or not the EL expression within the JSP page will be ignored.

b) What is EL? Explain immediate EL, deferred EL, LValue and RValue in detail. (5)

Expression Language (EL) is mechanism that simplifies the accessibility of the data stored in Java bean component and other object like request, session and application, etc.
Immediate Evaluation:
All expressions using the ${} syntax are evaluated immediately. These expressions can only be used within template text or as the value of a JSP tag attribute that can accept runtime expressions. The following example shows a tag whose value attribute references an immediate evaluation expression that gets the total price from the session-scoped bean named cart:
<fmt:formatNumber value="${sessionScope.cart.total}"/>
The JSP engine evaluates the expression, ${sessionScope.cart.total}, converts it,and passes the returned value to the tag handler.Immediate evaluation expressions are always read-only value expressions. The expression shown above can only get the total price from the cart bean; it cannot set the total price.

Deferred Evaluation:
Deferred evaluation expressions take the form #{expr} and can be evaluated at other phases of a page life cycle as defined by whatever technology is using the expression. In the case of JavaServer Faces technology, its controller can evaluate the expression at different phases of the life cycle depending on how the expression is being used in the page. The following example shows a JavaServer Faces inputText tag, which represents a text field component into which a user enters a value.The inputText tag's value attribute references a deferred evaluation expression that points to the name property of the customerbean.
<h:inputText id="name" value="#{customer.name}" />
For an initial request of the page containing this tag, the JavaServer Faces implementation evaluates the #{customer.name} expression during the render response phase of the life cycle. During this phase, the expression merely accesses the value of name from the customerbean, as is done in immediate evaluation.

c) Explain the advantages of JSTL over JSP. (5)
Advantages of JSTL over JSP:
1. Standard Tag: It provides a rich layer of the portable functionality of JSP pages. It's easy for a developer to understand the code.
2. Code Neat and Clean: As scriplets confuse developer, the usage of JSTL makes the code neat and clean. **3. Automatic JavabeansInterospection Support:** It has an advantage of JSTL over JSP scriptlets. JSTL Expression language handles JavaBean code very easily. We don't need to

downcast the objects, which has been retrieved as scoped attributes. Using JSP scriptlets code will be complicated, and JSTL has simplified that purpose.

4. Easier for humans to read: JSTL is based on XML, which is very similar to HTML. Hence, it is easy for the developers to understand.

5. Easier for computers to understand: Tools such as Dreamweaver and front page are generating more and more HTML code. HTML tools do a great job of formatting HTML code. The HTML code is mixed with the scriplet code. As JSTL is expressed as XML compliant tags, it is easy for HTML generation to parse the JSTL code within the document.

d) Explain JSTL core Tag Library. (5)

Tag Libraries

- A tag library is a set of elements that encapsulate specific application functionality, which are the used within the JSP pages.
- JSTL tag libraries provide a wide variety of such functionality, which can be broken down into specific functional areas belonging to an application.
- JSTL is composed of 5 tag libraries:
  - The Core Tag Library
  - The XML Tag Library
  - The Internationalization and Formatting Tag Library
  - The Database/SQL Tag Library
  - Functions Tag Library

The Core Tag Library

- It contains the tags which are essential to any web application. Eg: looping, evaluation of expression and basic input and output.
- It contains all the set of tags which are required for writing any JSP.
- The URI of the core tag library is: http://java.sun.com/jsp/jstl/core and the prefix is c.
- The core area comprises 4 distinct functional sections:
  - General purpose actions
  - Conditional actions or flow control statements
  - Iteration actions
  - URL related actions

Following table lists out the core JSTL Tags –

1.<c:out>: Like <%= ... >, but for expressions.
2.<c:set >: Sets the result of an expression evaluation in a 'scope'
3.<c:remove >: Removes a scoped variable (from a particular scope, if specified).
4.<c:catch>: Catches any Throwable that occurs in its body and optionally exposes it.
5.<c:if>: Simple conditional tag which evalutes its body if the supplied condition is true.
6.<c:choose>: Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <when> and <otherwise>.
7.<c:when>: Subtag of <choose> that includes its body if its condition evalutesto 'true'.
8.<c:otherwise >: Subtag of <choose> that follows the <when> tags and runs only if all of the prior conditions evaluated to 'false'.
9.<c:import>: Retrieves an absolute or relative URL and exposes its contents to either the page, a String in 'var', or a Reader in 'varReader'.
10.<c:forEach >: The basic iteration tag, accepting many different collection types and supporting subsetting and other functionality .

e) Explain the implicit objects of JSP. **(5)**

There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages.

- JSP implicit objects are created during the translation phase of JSP to the servlet
- These objects can be directly used in scriplets that goes in the service method.
- They are created by the container automatically, and they can be accessed using objects.

A list of the 9 implicit objects is given below:

1.Request

- Javax.servlet.HttpServletRequest
- It includes
  • String getParameter(String name)
  • Enumeration getParameterNames(),

2.Response

- HttpServletResponse
  • void addCookie(Cookie obj)
  • void sendRedirect(String Location)

3.Out

- Output Stream for content (JspWriter-buffered version of PrintWriter)
  • Its methods are:
  • void clear()
  • void close()

4.Session

- HttpSession
- Some of its methods are:
  • HttpSession getSession(boolean)
  • String getId()
  • long getCreationTime()

5. application:

- ServletContext
- Some of its methods are:
  • String getInitParameter(String name)
  • Enumeration getInitParameterNames()

6.page:

- Java.lang.Object
  • The page object represents the current JSP page itself and can be accessed using 'this' reference.

7.config:

- ServletConfig (configuration data) Some of
- its methods are:
  • String getServletName()
  • ServletContext getServletContext()

8. exception:

- Available on error pages (java.lang.Throwable)
  • The exception object available only in error page (jsp that has the attribute isErrorPage=true in the page directive.

9. pageContext -

- javax.servlet.jsp.PageContext
- Its method is:
  • HttpSession getSession()

f) Explain the various scope of JSP application. (5)

The availability of a JSP object for use from a particular place of the application is defined as the scope of that JSP object. Every object created in a JSP page will have a scope.

There are 4 scopes of JSP page:

1. Request
2. Page
3. Session
4. Application

- **Page:**
  'page' scope means, the JSP object can be accessed only from within the same page where it was created. The default scope for JSP objects created using tag is page. JSP implicit objects out, exception, response, pageContext, config and page have 'page' scope.
- **Request:**
  A JSP object created using the 'request' scope can be accessed from any pages that serves that request. More than one page can serve a single request. The JSP object will be bound to the request object. Implicit object request has the 'request' scope.
- **Session:**
  'session' scope means, the JSP object is accessible from pages that belong to the same session from where it was created. The JSP object that is created using the session scope is bound to the session object. Implicit object session has the 'session' scope.
- **Application:**
  A JSP object created using the 'application' scope can be accessed from any pages across the application. The JSP object is bound to the application object. Implicit object application has the 'application' scope.

Q4 Attempt any three of the following: (15)

a)Explain about enterprise bean container. (5)

An **Enterprise JavaBeans (EJB)** container provides a run-time environment for enterprise beans within the application server. The container handles all aspects of an enterprise bean's operation within the application server and acts as an intermediary between the user-written business logic within the bean and the rest of the application server environment. One or more EJB modules, each containing one or more enterprise beans, can be installed in a single container. The EJB container provides many services to the enterprise bean, including the following:

- Beginning, committing, and rolling back transactions as necessary.
- Maintaining pools of enterprise bean instances ready for incoming requests and moving these instances between the inactive pools and an active state, ensuring that threading conditions within the bean are satisfied.
- Most importantly, automatically synchronizing data in an entity bean's instance variables with corresponding data items stored in persistent storage.

By dynamically maintaining a set of active bean instances and synchronizing bean state with persistent storage when beans are moved into and out of active state, the container makes it possible for an application to manage many more bean instances than could otherwise simultaneously be held in the application server's memory. In this respect, an EJB container provides services similar to virtual memory within an operating system.

Following work is done by container:

- Allocates resources such as threads, sockets and database connections.
- Manages the life and death of enterprise beans.
- Converts the network less beans into distributed, network aware objects in order to service the remote clients.
- Serializes the requests when multiple clients call the methods on a bean's instance.
- Saves the conversational state of an object between 2 method calls.
- Gives security clearance to call a method on the bean.
- Manages the transaction of the enterprise bean.
- Maintains persistent data by synchronizing the data updates with beans and databases.
- Help registering and deploying components.

b) What are the different type of beans Explain **(5)**

EJB is primarily divided into three categories:

- Session Beans
- Message Driven Beans
- Entity Bean

Session Beans:

- It is a short-lived object that executes on behalf of a single client.
- It is a temporary, logical extension of a client application that runs on the application server.
- It can update data.

Types of Session Beans

- Stateless Session Bean
- Stateful Session Bean

Stateless session bean

It does not remember anything about the client between the calls of the method. It forgets about the client once the business process completes or executes. The bean's state has no data regarding a specific client. These are lightweight and provide a high degree of scalability. Since they do not retain the client state, it can be used across multiple clients.

Stateful session bean

It can remember conversations between the client of the application and the application itself across method calls. They store the data , when a client calls the method again the bean remembers the client's previous method call. Only a single client can use a stateful session bean at a time. The state is retained for the duration of the client bean session.

Message Driven Beans

- It allows enterprise applications to process messages asynchronously.
- Is an act as a JMS message listener, which is similar to an event listener except that it receives messages instead of events
- The messages may be sent by any component, an application client, another enterprise bean or a web component, a JMS application or system that does not use java EE technology.
- Message Driven Beans can consumes JMS messages from external entities and act accordingly.

Entity beans

Entity beans represent persistent data storage. User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean.

c) Explain the working behind message driven bean. (5)

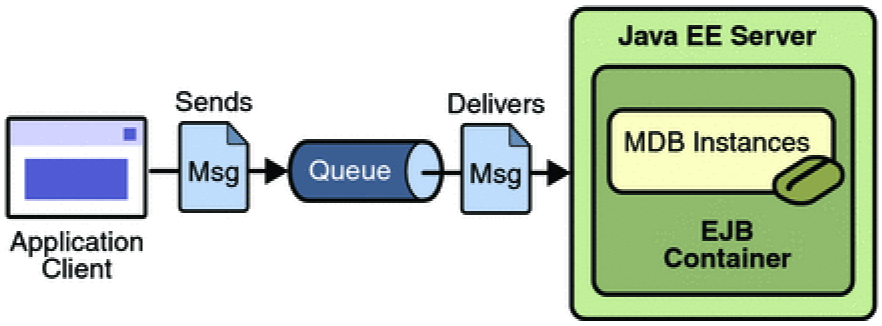
- A message driven bean (MDB) is a bean that contains business logic. But, it is invoked by passing the message. So, it is like JMS Receiver.
- MDB asynchronously receives the message and processes it.
- A message driven bean receives message from queue.
- JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication.
- JMS is also known as a messaging service
- A message driven bean is like stateless session bean that encapsulates the business logic and doesn't maintain state.

Understanding Messaging:

Messaging is a technique to communicate applications or software components. JMS is mainly used to send and receive message from one application to another.

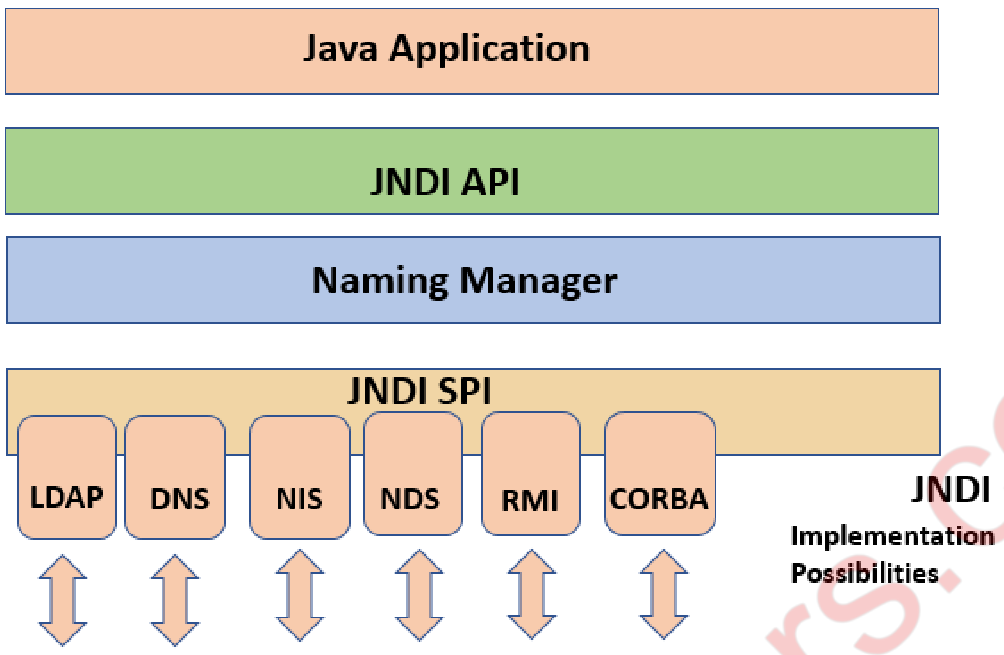
Requirement of JMS:

Generally, user sends message to application. But, if we want to send message from one application to another, we need to use JMS API.



d)Explain the concept of naming service. Add suitable illustration to it. (5)

- Enterprise application are created using multiple modules.
- These are the objects of that application.
- Each of these objects, when instantiated in memory must be bounded to a specific address that will permit it to instantly located, on demand.
- For handling to such objects are often referred to as references or pointers.
- Such pointers can be located to the memory of a single computer or be spread across the memory of multiple computers in a distributed environment.
- These are not in human readable form and frequently have communication protocols, platforms and current execution environment information bound to them.
- A name is simply a human readable logical value for referencing an application object as memory.
- Eg: a file name which provides a reference to a file object in a file naming system.
- An IP address that provides a reference to a host name in a DNS naming system.
- Java based applications use JNDI for naming and directory services. In context of EJB, there are two terms.
- **Binding** − This refers to assigning a name to an EJB object, which can be used later.
- **Lookup** − This refers to looking up and getting an object of EJB
- The JNDI architecture consists of an API and a service provider interface (SPI). Java applications use the JNDI API to access a variety of naming and directory services.
- The SPI enables a variety of naming and directory services to be plugged in transparently, thereby allowing the Java application using the JNDI API to access their services. See the following figure:



e)Explain basic look up in JNDI. Also explain resource injection in JNDI. (5)

Basic Lookup

- JNDI organizes its names into a hierarchy.
- A name can be any string such as org.ejb.ConverterBean or can also be an object that supports the Name interface.
- A string is the most common way to name an object.
- A name is bound to an object in the directory by storing either the object or a reference to the object in the directory service identified by the name.
- The user of the JNDI client API will first be required to create an initial context to the naming or directory service to connect to.
- This initial context establishes a connection with the naming or directory service when it is constructed with a set of properties that describes:
    - The specific naming or directory service provider library to use.
    - The URL of the naming or directory service process.
    - A user name and credentials
- JNDI API defines a context that specifies where to look for an object. The initial context is typically used as a starting point for all naming and directory operations.
- An initial context must be created using specific implementation and extra parameters required by the implementation. The initial context will be used to look up a name.

- After an initial context to a naming or directory service is established, the root context or sub-context reference is used, which performs various operations on the context depending on the type of context.
- Naming contexts can be used for binding and unbinding names and objects, list names and name bindings and lookup object references of named objects.

Resource injection

- One of the simplification features of Java EE is the implementation of basic Resource Injection to simplify web and EJB components.
- Resource injection enables you to inject any resource available in the JNDI namespace into any container-managed object, such as a servlet, an enterprise bean, or a managed bean. For eg, we can use resource injection to inject data sources, connectors, or any other desired resources available in the JNDI namespace.
- The type we'll use for the reference to the instance happen to be injected is usually an interface, which would decouple our code from the implementation of the resource. For better understanding of the above statement let's take a look at the example.
- The resource injection can be performed in the following three ways:
  - Field Injection
  - Method Injection
  - Class injection

f)Explain the lifecycle of an interceptor. **(5)**

To configure a life cycle callback interceptor method on an interceptor class, you must do the following:

1. Create an interceptor class.
   This can be any POJO class.
2. Implement the life cycle callback interceptor method.
   Callback methods defined on a bean's interceptor class have the following signature:

   Object <METHOD>(InvocationContext)

3 Associate a life cycle event with the callback interceptor method

A life cycle event can only be associated with one callback interceptor method, but a life cycle callback interceptor method may be used to interpose on multiple callback events. For example, @PostConstruct and @PreDestroymay appear only once in an interceptor class, but you may associate both @PostConstruct and @PreDestroy with the same callback interceptor method.

4 Associate the interceptor class with your EJB 3.0 session bean.

Using Annotations:

You can specify an interceptor class method as an EJB 3.0 session bean life cycle callback method using any of the following annotations:

• @PostConstruct

• @PreDestroy

• @PrePassivate (stateful session beans only)

• @PostActivate (stateful session beans only)

Q5 Attempt any three of the following: (15)

a) Explain the persistent standards available in java. (5)

- Most business applications require that data must be persistent.
- Data can be labeled as persistent only when it manages to survive problems such as system crashes and network failures.
- As multiple users request for data simultaneously, there is a definite possibility of data getting corrupted, if mid-request, system failure, occurs.
- Maintaining the persistence of data in an enterprise wide application is quite a challenging job.
- In enterprise application architecture, data persistence is implemented as:
    - Having data stored outside an application's active memory, known as persistent data store, typically a relational database or an object database or a file system.
    - Having a rollback system, where in case of system failure, the state of the data is rolled back to its last known valid data state.
- Persistence is one of the fundamental concepts of application development.
- Majority of applications use persistent data.
- Hence it is important to choose an appropriate persistence data store.
- While choosing the persistence data store the following fundamental pts are considered:
    - The length of time data must be persisted
    - The volume of data

JPA Versions

The first version of Java Persistence API, JPA 1.0 was released in 2006 as a part of EJB 3.0 specification. Following are the other development versions released under JPA specification: -

- JPA 2.0 - This version was released in the last of 2009. Following are the important features of this version: -
    - It supports validation.
    - It expands the functionality of object-relational mapping.
    - It shares the object of cache support.
- JPA 2.1 - The JPA 2.1 was released in 2013 with the following features: -
    - It allows fetching of objects.
    - It provides support for criteria update/delete.
    - It generates schema.
- JPA 2.2 - The JPA 2.2 was released as a development of maintainenece in 2017. Some of its important feature are: -
    - It supports Java 8 Date and Time.
    - It provides @Repeatable annotation that can be used when we want to apply the same annotations to a declaration or type use.
    - It allows JPA annotation to be used in meta-annotations.

b)Draw and explain the architecture of hibernate framework (5)

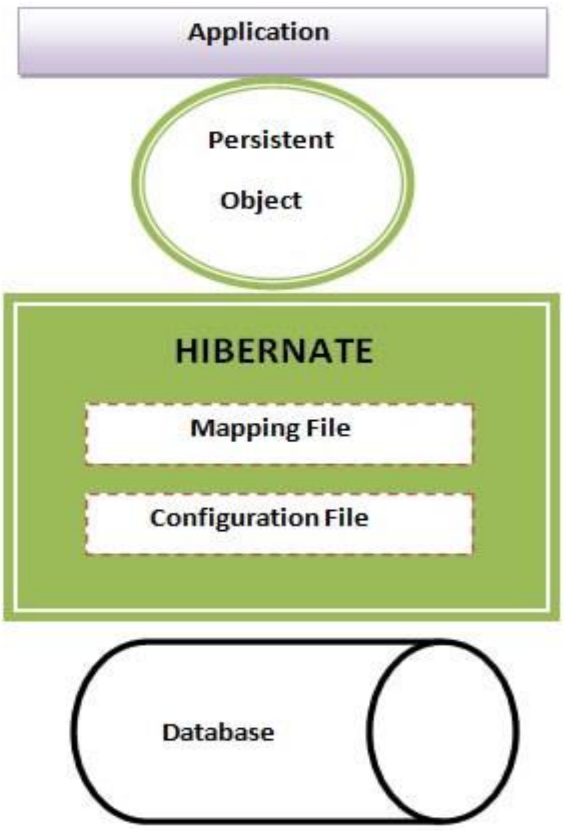
Hibernate:

- It is the latest open source persistence technology.
- It provides a framework for mapping an object oriented domain model to a traditional relational database.
- It was developed with a goal to relieve developers from 95% of common data persistence related programming tasks by:
    - Making developers feel as if the database contains plain java objects, without having them worry about how to get them out of / or back into database tables.
    - Allowing developers to focus on the objects & features of the application, without having worry about how to store them or find them later.
- Its primary feature is mapping from:
    - Java classes to database tables
    - Java data types to SQL data types
    -

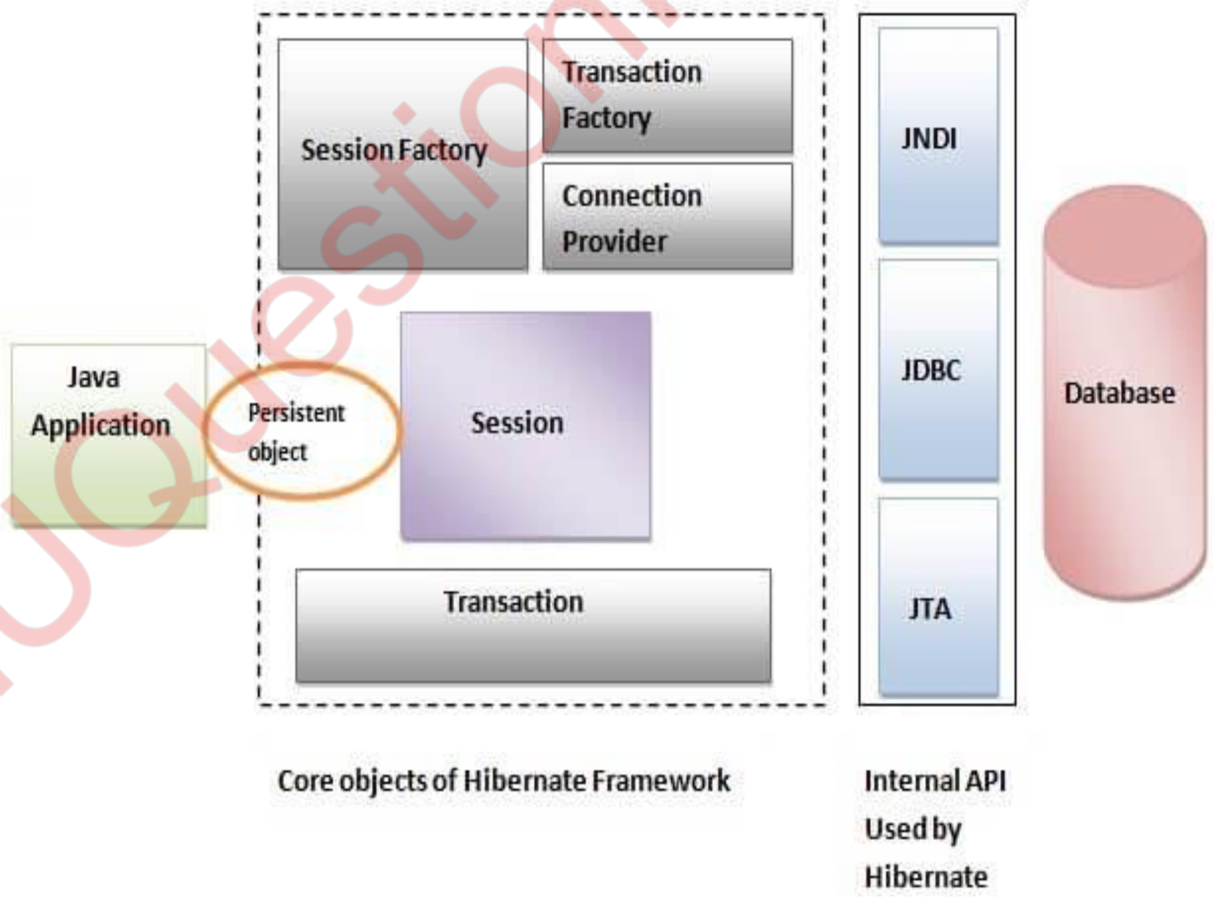
Architecture of Hibernate:

- **Configuration Object**: represents a configuration or properties file for hibernate.it is created once during application initialization. It reads the properties to establish a database connection. It is produced to create a session factory.
- **Session Factory**: is created with the help of a configuration object during the application start up. It serves as a factory for producing session objects when required. It is created once and kept alive for later use.
- **Session**: are lightweight and inexpensive to create. They provide the main interface to perform actual database operations. All the POJOs are saved & retrieved with the help of a session object.
- **Transaction**: represents a unit of work with the database. Any kind of modifications initiated via the session object are placed with in a transaction. Session object helps creating a transaction object. Which is used for a short time and are closed by either committing or rejecting.
- **Query**: persistence objects are retrieved using a query object. It allow using SQL or HQL queries to retrieve the actual data from the database and create objects.
- **Criteria**: persistence objects can also be retrieved using a criteria object. It uses an object/method based means of constructing and executing a request to retrieve objects.

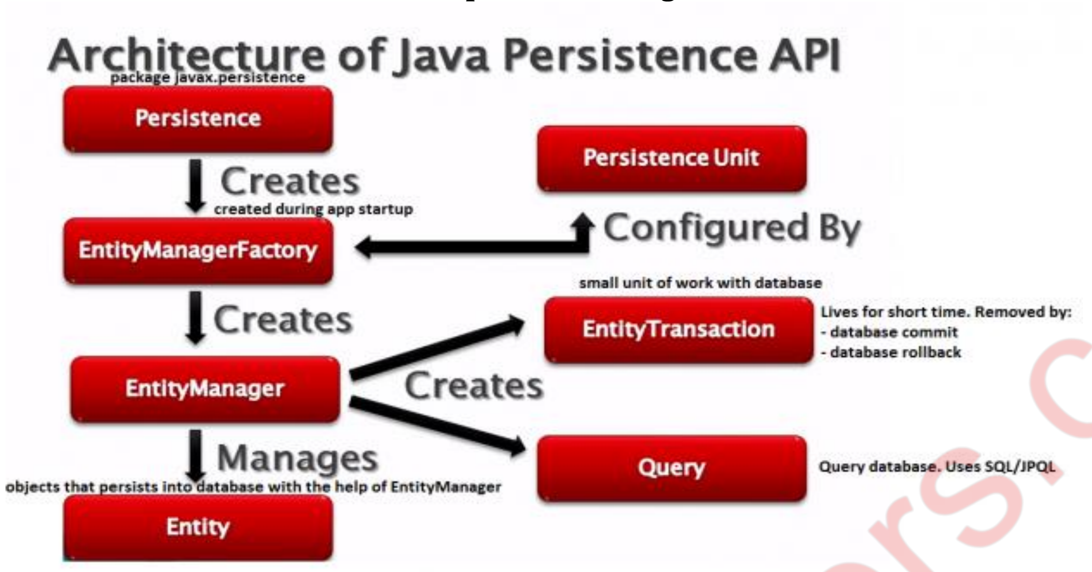
Let's see the diagram of hibernate architecture:



This is the high level architecture of Hibernate with



c) Explain JPA Architecture with the help of a neat diagram. **(5)**


Architecture of Java Persistence API
package javax.persistence
Persistence
Creates
created during app startup
EntityManagerFactory
Persistence Unit
Configured By
Creates
EntityManager
small unit of work with database
EntityTransaction
Lives for short time. Removed by:
- database commit
- database rollback
Creates
Manages
objects that persists into database with the help of EntityManager
Query
Query database. Uses SQL/JPQL
Entity

Persistence:

- The javax.persistence.Persistence class contains static helper methods to obtain EntityManagerFactory instances in a vendor-neutral fashion.

EntityManagerFactory:

- It is created with the help of a persistence unit during the application start up. It serves as a factory for spawning EntityManager objects when required. Typically it is created once and kept alive for later use.

EntityManager:

- The EntityManager object is lightweight and inexpensive to create. It provides the main interface to perform actual database operations.
- All the persistence objects are saved and retrieved with the help of an EntityManager object. These are created as needed and destroyed when not required.

Entity:

- Entities are persistence objects that represent datastore records.

EntityTransaction:

- A transaction represents a unit of work with the database. Any kind of modification initiated via EntityManager object are placed within a transaction. An EntityManager object helps creating an EntityTransaction object. These objects are typically used for a short time and are closed by either committing or rejecting.

Query:

- Persistence objects are retrieved using a query object. It allows using SQL or JPQL queries to retrieve the actual data from the database and create objects.

Criteria:

- Criteria API is a non-string-based API for the dynamic construction of object-based queries.
- Criteria query objects are passed to the EntityManager's createQuery() method to create query objects and then executed using the methods of Query API.

The above classes and interfaces are used for storing entities into a database as a record. They help programmers by reducing their efforts to write codes for storing data into a database so that they can concentrate on more important activities such as writing codes for mapping the classes with database tables.

d) What is Impedance Mismatch? How it can be solved? (5)

First problem, what if we need to modify the design of our database after having developed a few pages or our application? Second, loading and storing objects in a relational database exposes us to the following five mismatch problems –

1 Granularity:

Sometimes you will have an object model, which has more classes than the number of corresponding tables in the database.

2 Inheritance:

RDBMSs do not define anything similar to Inheritance, which is a natural paradigm in object-oriented programming languages.

3 Identity:

An RDBMS defines exactly one notion of 'sameness': the primary key. Java, however, defines both object identity (a==b) and object equality (a.equals(b)).

4 Associations:

Object-oriented languages represent associations using object references whereas an RDBMS represents an association as a foreign key column.

5 Navigation:

The ways you access objects in Java and in RDBMS are fundamentally different.

The **Object-Relational Mapping** (ORM) is the solution to handle all the above impedance mismatches

ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java, C#, etc. An ORM system has the following advantages over plain JDBC –

- Let’s business code access objects rather than DB tables.
- Hides details of SQL queries from OO logic.
- Based on JDBC 'under the hood.'
- No need to deal with the database implementation.
- Entities based on business concepts rather than database structure.
- Transaction management and automatic key generation.
- Fast development of application.

e) Explain different components of hibernate. (5)

Components of Hibernate:

- **Connection Management**: provides efficient management of the database connection. Database connection is the most expensive part of an application that allows interacting with the database.
- **Transaction Management**: provides ability to execute more than one database statements at a time.
- **Object Relational Mapping**: is a technique of mapping the data representation from an object model to a relational model. This part is used to perform CRUD operations from the underlying database tables.

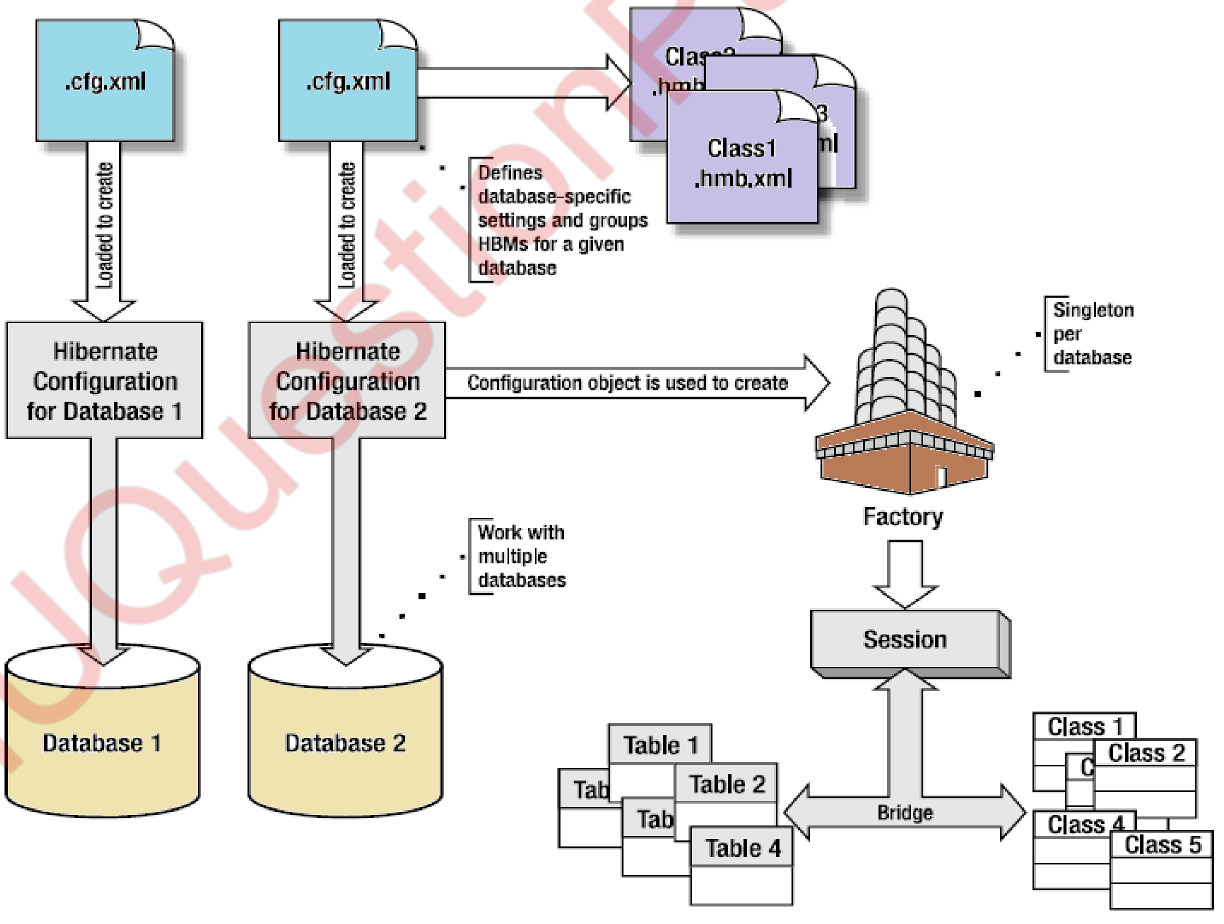
Core components of Hibernate

Below are the important elements of Hibernate

- hibernate.cfg.xml: This file has database connection details
- hbm.xml or Annotation: Defines the database table mapping with POJO. Also defines the relation between tables in java way.
- **SessionFactory:**
  - There will be a session factory per database.
  - The SessionFacory is built once at start-up o It is a thread safe class
  - SessionFactory will create a new Session object when requested
- **Session:**
  - The Session object will get physical connection to the database.
  - Session is the Java object used for any DB operations.
  - Session is not thread safe. Hence do not share hibernate session between threads
  - Session represents unit of work with database
  - Session should be closed once the task is complete

f) Explain the modus operandi behind hibernate application. **(5)**

- Hibernate is an Object-Relational Mapping (ORM) solution for JAVA. It is an open source persistent framework created by Gavin King in 2001. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application.
- Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieves the developer from 95% of common data persistence related programming tasks.
- Hibernate sits between traditional Java objects and database server to handle all the works in persisting those objects based on the appropriate O/R mechanisms and patterns.



.cfg.xml
Loaded to create
.cfg.xml
Loaded to create
Class1 .hmb.xml
Defines database-specific settings and groups HBMs for a given database
Hibernate Configuration for Database 1
Hibernate Configuration for Database 2
Configuration object is used to create
Singleton per database
Factory
Work with multiple databases
Session
Database 1
Database 2
Table 1
Table 2
Table 4
Bridge
Class 1
Class 2
Class 4
Class 5

How Hibernate works java-latte.blogspot.in